

TDB-ACC-NO: NN9801281

DISCLOSURE TITLE: Improved Process for Visual Development
of Client/Server Programs

PUBLICATION-DATA: IBM Technical Disclosure Bulletin,
January 1998, US

VOLUME NUMBER: 41

ISSUE NUMBER: 1

PAGE NUMBER: 281 - 284

PUBLICATION-DATE: January 1, 1998 (19980101)

CROSS REFERENCE: 0018-8689-41-1-281

DISCLOSURE TEXT:

JavaBeans is quickly accepted as a promising component model. IBM* Visual Age for Java provides various tools supporting beans development. One portion of the tools is the support for developing enterprise Java bean applications. The tool allows the user to visually develop distributed beans applications without writing communication code.

The Visual Age approach is to generate the proxy beans from users designated server beans and then link the generated proxy beans with the client beans in a visual environment. However, this approach has two notable drawbacks. First, the users have to deal with the system generated classes which they may not be comfortable with. In

addition,
there are a large number of the generated classes. For
example,
there
are eight generated classes and two generated
interfaces in the
simple
example provided with Visual Age for Java. Second, the
users have to
link the client proxy bean first and then test the
correctness of
their
program. If there are something wrong, the users have
to repeat the
whole development process.

The following process is proposed to allow a
Java bean
application built in a visual environment and run
remotely without
the
above two problems.

The improved process can be summarized as
follows. Users link
their client bean with the server bean first in their
visual
environment
such as Visual Age.

A tool can then generate the client and the
server
proxy beans and allow the client bean to interact with
the client
proxy
bean, and let client proxy bean talk with the server
proxy bean using
such as Remote Method Invocation (RMI). The user beans
will not need
to be changed. Next described is one approach of
constructing this
proxy generation tool.

The tool for generating the proxy beans can be
built by using
Javabeans' introspection and Java's reflection
facilities.

For
example,
in CR-xxx, we described a process to dynamically split

a Java program
 and run it as a client/server application, which
 includes a process
 of
 generating proxy classes using reflection. For
 JavaBeans, the
 introspection is needed if users do not use the design
 pattern in the
 event handling code. Moreover, the following two
 problems must be
 solved in generating proxy bean classes: the name
 conflicts and event
 handling. The following examples are used to describe
 the problems
 and
 this solution.

Assume that there are two beans X and Y.
 The class Y and its
 objects will be placed in a remote machine and the
 class X and its
 objects are in the local machine. X is a listener to a
 property
 propY in Y. When X is notified the changes of propY,
 it will call a
 method B defined in Y.

```

public Class X implements
java.beans.PropertyChangeListener [
    private Y y = new Y();
    // register as a listener
    public X() [
        y.addPropertyChangeListener(this);

    ]
    // handle events
    public void
propertyChange(java.beans.PropertyChangeEvent evt) [
    if ((evt.getSource() == y) &&
        (evt.getPropertyName().equals("propY"))) [
        y.B();
    ]
    ] ]
public Class Y [
    private String propY = "propY"; // property to
check
    private java.beans.PropertyChangeSupport pcs;

```

```

        public Y() [
            pcs = new java.beans.PropertyChangeSupport(this);
        ]
        // Pattern methods for registering a listener
        public void addPropertyChangeListener
            (java.beans.PropertyChangeListener pcl) [
                pcs.addPropertyChangeListener (pcl);
            ]
        // Getter and setter methods
        public String getPropertyY() [
            return propY;
        ]

        public void setPropertyY(String newValue) [
            pcs.firePropertyChange("propY", oldValue,
newValue);
        ]
        // Code to change the property
        public void changeProperty() [
            setPropertyY(newValue);
        ] ]

```

Name conflicts. Since the client bean X will talk with the client proxy bean Y' and we will not change the client bean, Y' needs to have the same class name as Y. This name conflict can be solve by storing Y' in a different directory.

In the runtime, the same named client proxy bean is running in the client machine. Therefore, the name conflict problem is solved. In Visual Age for Java, the client proxy bean has a different name as the server bean, which introduces another complication for users (the approach described here is very similar with the process used in CR-xxx.)

Event handling. In the case of X and Y in the same JVM, when propY changed in the object Y, an event will be sent to the corresponding object X which was registered as a listener of the property propY. However, now the object Y is running in the remote machine,

the event can not be sent to the object X in the normal way.

This problem is solved by the following approach. The client proxy Y' is designated as the source of the event which the client object is listen to. The server proxy Y'' is designated as the target object of the event from the server object. In other words, it listens the event sent by the server object. When the event is sent to the server proxy object, it sends a remote method call to the client proxy with the event object as a parameter.

The client proxy will use the value in the event object to fire the event and trigger the client bean. The client bean can then take the appropriate actions. For example, it may even invoke the method B in Y. This method invocation will be relayed by the client proxy Y' to call the remote method in the server proxy bean Y''.

One more detail was left on how the server proxy object to find out the client proxy object which will receive the remote method call when the event is received by the server proxy object. One solution would be to register the client proxy object with the server proxy object.

When the server proxy object receives the event, it will call back to the corresponding client proxy object. In summary, the key idea is the use of the proxy beans as new event

source and
target,
and pass event objects using the remote method calls
coded in the
proxy
beans.

As mentioned, this approach has two advantages.

First,
since the user's applications can be first assembled as
a single
application that is achieved: test beans locally, and
run beans
remotely. Therefore, it reduces the overhead of the
application
development. Second, the process of developing
distributed beans
applications becomes more transparent to the end user.
Now, users
need not to deal with large amount of beans generated
from the
systems.

This approach can be rapidly incorporated into
Visual Age for
Java. It can also be used as a value-added component
for other
visual
program development tools such as IBM's BeanMachine and
WebRunner,
Symantec Visual Cafe and Sun Java Workshop.

Symantec Visual Cafe contains a visual
development environment
to allow users to develop GUI component visually. It
can be used to
generate and assemble beans. Similarly, Sun Java
Workshop is a
visual
Java development environment which can generate beans.

Borland
JBuilder
includes full support for visually constructing beans,
applets, and
applications using standard and third-party beans.

Lotus BeanMachine
for
Java is an interactive visual authoring tool which
allows
non-programmers
and Web professionals to combine Java and JavaBeans

components into
intranet and Internet applications. It contains the
full JDBC
support
for open data access from text files, spreadsheets, and
relational
databases. IBM WebRunner contains bean tools to assist
in building
beans
visually.

Even though some of these products produce
limited support for
developing distributed applications, for example, Sun
Java WorkShop
allows the use of RMIC from the IDE, Borland JBuilder
will support
developing distributed applications, none of them
support automatic
visual development of distributed applications.

SECURITY: Use, copying and distribution of this data is
subject to the
restrictions in the Agreement For IBM TDB Database and
Related Computer
Databases. Unpublished - all rights reserved under the
Copyright Laws of the
United States. Contains confidential commercial information
of IBM exempt
from FOIA disclosure per 5 U.S.C. 552(b)(4) and protected
under the Trade
Secrets Act, 18 U.S.C. 1905.

COPYRIGHT STATEMENT: The text of this article is
Copyrighted (c) IBM
Corporation 1998. All rights reserved.